**ISULPAR (INSTITUTO SUPERIOR DO LITORAL DO PARANÁ**

**Disciplina: Segurança da Informação**

**Aluno: Diego A. Nunes**

**Professor: Luiz Efigênio**

**Data: 05/09/2025**

**Relatório – SQL Injection e Prevenção no Sistema de Login (Node.js + SQLite)**

**1. Introdução**

O SQL Injection é uma das vulnerabilidades mais comuns em aplicações web, explorando falhas de validação de entradas do usuário.  
Segundo Barreto et al. (2018) e Moraes (2021), essa falha pode comprometer a Confidencialidade, Integridade e Disponibilidade (CIA) da informação.

Neste relatório, foi desenvolvido um sistema simples de login e cadastro utilizando Node.js, Express e SQLite, com o objetivo de demonstrar:

* Como uma aplicação vulnerável poderia ser explorada.
* Como mitigar o ataque utilizando prepared statements.

**2. Metodologia**

O ambiente foi implementado com:

* **Backend:** Node.js + Express + SQLite.
* **Frontend:** HTML, CSS e JavaScript (fetch API).
* **Funcionalidades:** Cadastro de usuário e Login.

O código foi dividido em:

* app.js: servidor Node.js com rotas /register e /login.
* public/: contendo index.html, style.css e script.js.

**Observação:** Para o teste, foi criado um usuário de exemplo no banco de dados.

**3. Demonstração da Vulnerabilidade**

**3.1 Código vulnerável (hipotético)**

Um sistema inseguro poderia ser escrito assim:

// ❌ Inseguro – concatenação de strings

db.get(`SELECT \* FROM users WHERE username = '${username}' AND password = '${password}'`, (err, row) => {

...

});

**3.2 Exploração**

Se o atacante informar no campo **Usuário**:

' OR '1'='1

E qualquer senha no campo **Senha**, a query gerada seria:

SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = 'abc';

➡️ O trecho '1'='1' sempre retorna verdadeiro, permitindo **acesso sem credenciais válidas**.

📸 **Print 1** – Tentativa de login com SQL Injection.  
📸 **Print 2** – Sistema aceitando acesso (em versão vulnerável).

**4. Mitigação Implementada**

**4.1 Código seguro**

No sistema desenvolvido, foi utilizado **prepared statement** com placeholders ?:

// ✅ Seguro – query parametrizada

db.get("SELECT \* FROM users WHERE username = ?", [username], async (err, row) => {

...

});

Dessa forma, mesmo que o usuário tente inserir ' OR '1'='1, o SQLite tratará esse valor como **string literal**, e não como comando SQL.

**5. Teste de Mitigação**

**5.1 Tentativa de ataque**

Campos preenchidos no formulário:

* Usuário: ' OR '1'='1
* Senha: teste123

**5.2 Resultado**

O sistema retornou: **“Usuário não encontrado”**.  
O login falhou porque não existe usuário registrado com esse nome literal.

📸 **Print 3** – Tentativa de login com ' OR '1'='1.  
📸 **Print 4** – Resposta do sistema (falha no login).

**6. Estrutura do Projeto**

**O projeto foi organizado em pastas e arquivos da seguinte forma:**

**SQL/ # Pasta raiz do projeto**

**│── app.js # Servidor Node.js com rotas de login e cadastro**

**│── package.json # Configurações do projeto e dependências**

**│── users.db # Banco de dados SQLite (gerado automaticamente)**

**└── public/ # Arquivos estáticos (frontend)**

**│── index.html # Página principal (formulário de login/cadastro)**

**│── style.css # Arquivo de estilos (design do formulário)**

**└── script.js # Lógica frontend (tabs + requisições via fetch)**

**⚙️ Tecnologias Utilizadas**

* **Node.js → servidor backend.**
* **Express → framework web para rotas e middlewares.**
* **SQLite → banco de dados leve para persistência.**
* **Bcrypt → para hash seguro de senhas.**
* **HTML + CSS + JS → interface de login e cadastro.**

**LINK:**

**7. Conclusão**

O experimento comprovou que:

* Um sistema que utiliza **concatenação de strings** nas queries SQL está vulnerável a **SQL Injection**.
* A utilização de **prepared statements** (consultas parametrizadas) é a forma correta de evitar essa exploração.
* O sistema implementado em **Node.js + SQLite** demonstrou-se **seguro contra SQL Injection**, rejeitando tentativas de login maliciosas.

Esse exercício destaca a importância de aplicar **boas práticas de programação segura**, protegendo sistemas contra ataques comuns.

**8. Referências**

* BARRETO, Jeanine dos Santos et al. *Fundamentos de segurança da informação.* Porto Alegre: SAGAH, 2018.
* MORAES, Alexandre Fernandes de. *Cibersegurança e a nova geração de firewalls.* São Paulo: Saraiva, 2021.
* WANDERLEY, Alex Rodrigo Moisés Costa; PONTUAL, Ricardo de Almeida. *Gerenciamento de servidores.* São Paulo: Saraiva, 2019.